ISSN 0265-2919

80p 24

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO

An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

- **A computer, monitor and disk drives for £500? We examine the Tatung Einstein.**
- **Out with a bang. We come to the conclusion of the BBC Minefield project.**
- **Soft sounds from your computer—we begin a new series on computerised music.**

QUIZ

1) Having taken the trouble to assemble a program, why should one wish to disassemble it again?

2) What is the purpose of 'top-down' design?

3) Which type of disk is used on the Macintosh?

4) What is the significance of ! (exclamation mark) in the Towers of Hanoi program?

Answers To Last Week's Quiz

A1) True. Using the RAM within the Psion Organiser means that data can be deleted and the memory reused. However, the memory in the datapaks cannot be reused.

A2) There is no edge connector on the Electron, so if Acorn plan any further interfaces they need a connector for the Plus 1.

A3) Audiogenic's Alice In Videoland uses 90 Kbytes of code.

A4) Yes. Although the Spectrum version uses a GOTO statement to control the keyboard, the program is designed as discrete subroutines.

QUIZ

COVER PHOTOGRAPHY BY IAN McKINNELL

Editor Jim Lennox; **Managing Editor** Mike Wesley; **Art Director** David Whelan; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Art Editor** Claudia Zeff; **Chief Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Sub Editor** Steve Mann; **Researcher** Melanie Davis; **Contributors** Steve Colwill, Steve Malone, Peter Jackson, Ted Ball, Richard Pawson, Martha Ellen Zenfell, Graham Storrs; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Controller** Peter Taylor-Medhurst; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

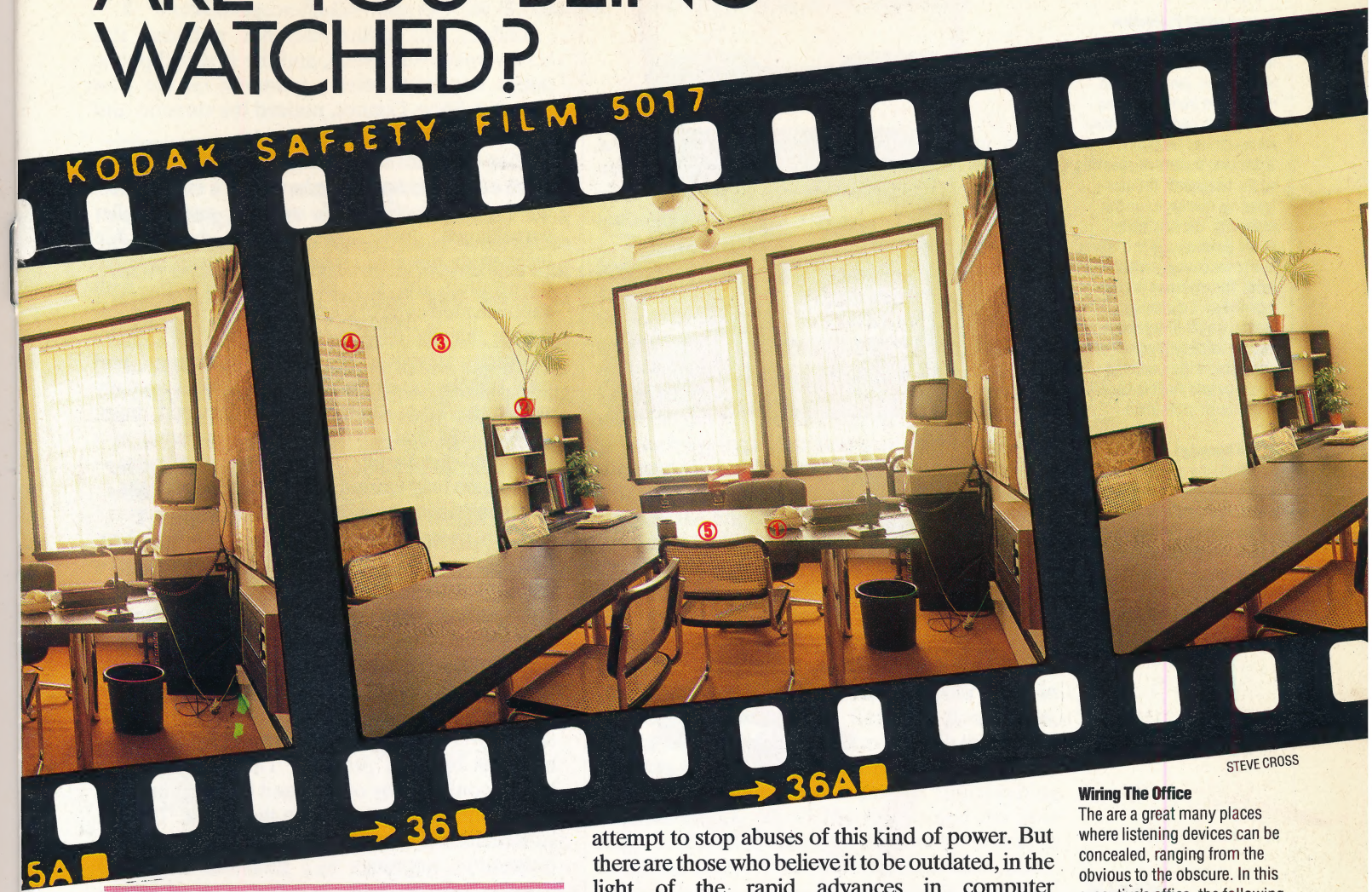
How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.

Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

ARE YOU BEING WATCHED?



STEVE CROSS

While the world's law enforcement agencies swoop on teenage 'hackers' for cracking computer codes, those same agencies, and those without the backing of the law, are involved in similar surreptitious activities. In this article we look at the latest developments in computer surveillance techniques.

The official use of computer surveillance is constantly expanding into more and more areas of everyday life, and the odds are that we have all found our way into some system at some time.

Some of the surveillance is innocent enough and of vital importance to government agencies, such as the records of cars and licences held on the DVLC computers at Swansea or the DHSS's computer-based social security records. The possibility has now been raised of linking these various systems together, to correlate the files from the DVLC and DHSS systems with the files on, say, the Police National Computer at Hendon. That gives the authorities much more power to monitor the activities of the entire population.

The Data Protection Act was introduced in an attempt to stop abuses of this kind of power. But there are those who believe it to be outdated, in the light of the rapid advances in computer technology since its introduction.

A lot of the techniques used in computer surveillance and security systems involve pattern recognition, a technique whereby the computer compares what it 'sees' with patterns already stored in memory. The drawback to pattern recognition is that it requires large amounts of memory space and vast amounts of computer processing power. Now both of those are available, and cheap, which has enabled significant advances to be made.

An example is the new fingerprinting system that Logica has installed for the Metropolitan Police at London's New Scotland Yard. It has taken 15 years of development to create the system, which can store 650,000 fingerprints and 100,000 'marks' — partial prints found at the scene of a crime. The system simply compares the marks with all of the stored prints, to see if they match with anything on file. This application needs the computer power of Prime minicomputers, in conjunction with highly efficient array processors and high-performance television monitors and cameras. Even so, it can only check 200 or 300 marks against the 650,000

Wiring The Office

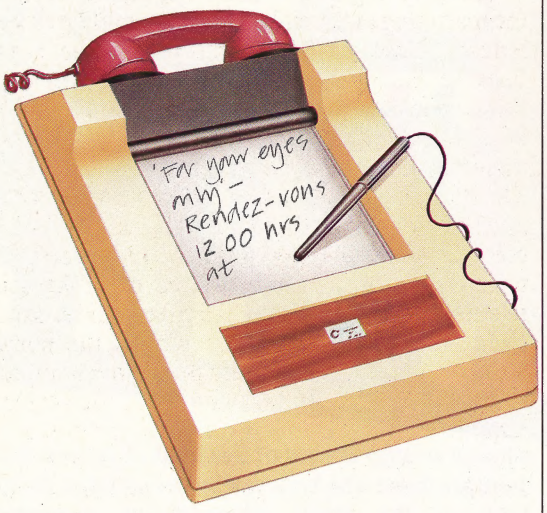
The are a great many places where listening devices can be concealed, ranging from the obvious to the obscure. In this executive's office, the following places could be concealing 'bugs':

1) The telephone. The bug could be in the mouthpiece or in the body of the phone, or in the lamp, which is placed very near.
2) Potted Plant. In the earth, under the pot, or even disguised as a real bug!
3) The wall. Bugs have been known to be embedded in the wall as 'studs', or behind cork panelling.
4) Hanging picture. Disguised as a support hook or concealed behind.
5) In the desk. Underneath the desk-top or in any of the drawers

Tools Of The Trade

Computerised Protection
Many executives, and people who deal with classified information, are protecting themselves from electronic spying with gadgets like this computerised telephone scrambler. Instead of speaking on the telephone, the user types his messages on the keyboard. The scrambler sends the message in a 'silent' form over ordinary telephone lines. The message is then received on a matching system's built-in screen. A built-in voice synthesiser can also convert the incoming message into audible speech at the touch of a button

Coded Message
Similar to the computerised scrambler, this system sends a scrambled message from a handwritten note. The scrambled message is protected from eavesdropping and bugging because it is sent silently. Even signatures can be sent in this way



Stress Analyser
The voice stress analyser measures the amount of stress in the voice and displays its results instantly in a simple numeric readout. This is really a sophisticated lie detector and can also indicate whether the person is anxious or tense

stored prints per day.

A similar pattern recognition and comparison system is mounted on a bridge across the M1 motorway, with cameras pointing down at each lane. This system actually captures pictures of the number plates of approaching cars, then uses computer power to analyse the pictures and check the numbers against a file of wanted cars. The information that one of the wanted vehicles has been seen can then be radioed direct to motorway patrol vehicles who will intercept the car.

Initially, the police did not publicise this achievement very much, and the first real public notice was taken after a journalist on the *New Scientist*, Steve Connor, noticed the cameras and asked what they were for.

One obvious area in which developments in microcomputer hardware have had a major effect is the production of smaller and smaller surveillance devices — that is, bugs. Chip technology has made it possible to produce radio transmitters the size of a grain of rice, with sophisticated control electronics built in. A typical device 'bleeds' its power from the Post Office's electricity supply to the bugged telephone, and only switches itself on when someone is actually speaking. Then there are self-powered bugs, equally tiny, that are dropped in the corner of a room and pick up all conversation in that room — once again only working when someone speaks — for transmission to a distant receiver.

Even more in the style of James Bond, there is a 'distant' bug that fires a laser beam at a window. The vibrations of the glass caused by conversation are picked up as interference in the reflected laser beam, and the speech information is retrieved from this interference — by computer, of course.

In military operations, as opposed to undercover security work, the computer operators have the opposite problem. They aim to avoid being monitored by others, and once again chips and computers have come to the rescue. Today's battlefield radio transmitters and receivers use frequency hopping — processor-controlled jumping from frequency to frequency according to a preset code — to avoid eavesdropping and jamming.

Computers in surveillance and security are, at the moment, big machines of the type and power used for complex code-cracking at places like the CIA and the National Security Agency in the US — not to mention Britain's MI5 and MI6. But the advances in hardware technology mean that fingerprint recognition — perhaps even face recognition — will soon be automated and made very inexpensive.

In the future, it is possible that police cars will be equipped with onboard computers that could instantly pull out data from a school record, criminal record, medical record, social security record or any other official file. All this would be accessed by sliding a plastic national insurance card into a slot in the computer. The machine would accept fingerprints and a photograph, compare them with central files, and make sure of the suspect's identity.

This might seem a paranoid vision, but the technology is there, or almost there, to do this now, and there are people in the law and order lobby who would like to see it done.

THE GENTLE TOUCH

Mastering the skill of touch-typing is essential for many aspects of home computing. What was once regarded as a clerical chore has in recent years been taken up by programmers as well as journalists and typesetters; in fact, anyone who needs to type quickly and accurately.

The ability to type complete words on paper or for viewing on-screen, rather than laboriously tapping at individual letters on a keyboard, eliminates a time-consuming step in a process where speed is often of importance, and reduces the margin for error.

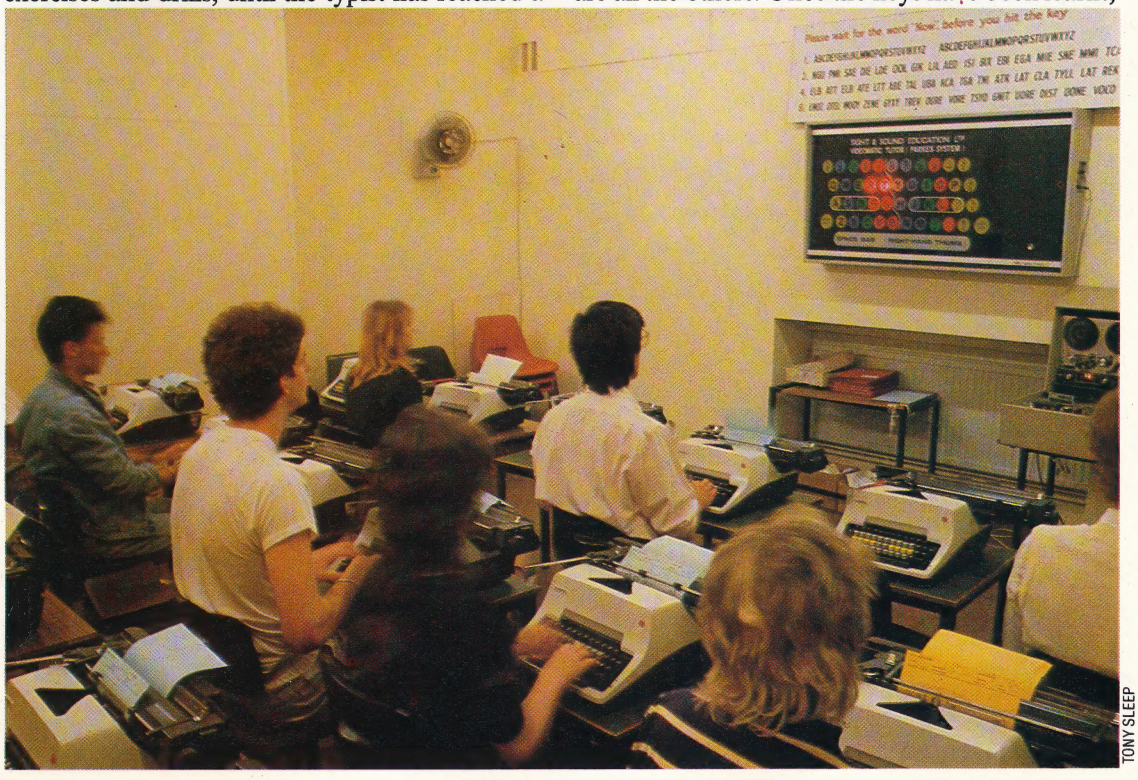
Typing has traditionally been taught through instruction manuals, or in a classroom, using a variety of teaching methods. There is even a range of software programs on the market that links lessons directly to the computer. Regardless of the teaching method, there are four principles that must be learned to become an accomplished touch-typist. These are:

- Mastering the keyboard
- Training the eyes on a screen (or paper)
- Accuracy
- Speed

These skills are developed slowly, through exercises and drills, until the typist has reached a level of proficiency. Constant repetition is the main method of learning; striking the same letters until the sequence of fingering becomes automatic.

The standard English language typewriter layout is known as the QWERTY keyboard, so called because these are the letters displayed on the left-hand side of the second of the four rows of keys. Keyboard symbols and letters are positioned according to their frequency of use, and each keyboard row is divided into left- and right-hand sides for the purpose of instruction.

Touch-typing instruction generally begins with the mastering of the eight keys in the middle row of letters. These are known as the 'home keys', because the fingers return to these keys once they have typed other keys on the keyboard. The home keys for the left side of the typewriter, and therefore the four fingers of the left hand, are asdf. The home keys for the right side of the typewriter, and therefore for the right hand, are jkl; (the semi-colon). Once these keys have been mastered, the typist learns the location of the other keys in relation to their position from the home keys, feeling and stretching for them while in the 'home' position. One surprise for the newcomer to typing is the use the little finger is put to. Touch-typing requires the use of *all* the fingers and the little finger is allocated a number of keys to 'cover', as are all the others. Once the keys have been learnt,



TONY SLEEP

On Location
Sight and Sound offer touch-typing courses for beginners and have a proven record of success. In this picture, students are being taught by a combination of audio and visual techniques

the typist moves on to the space bar (pressed with the thumbs), and the shift key for capital letters and for keys with two characters allocated to them.

Unfortunately for the home computer user, the correct use of number keys is rarely if ever touched upon in touch-typing courses or manuals. For the programmer, especially, the number keys form a vital and integral part of the computer keyboard. Touch-typing of numbers is fairly easy to learn once the concept and operation of the home keys have been mastered. The fingers of the left hand simply extend above the alphabetical keys to take in the numerals one to six, while the fingers of the right hand are responsible for seven to nought, and any following symbol keys.

An interesting extension of the 'home key' approach can be made when using a home computer with graphics characters that can be accessed directly from the keyboard. By learning the location of the graphics symbols, it should be possible to incorporate them into your touch-typing programme.

Training the eyes on a screen (or paper) is learned at the same time as mastering the keyboard. This is the most important aspect of touch-typing, setting it apart from 'two finger tapping' or the 'look and search' approach to typing. Covering the keys with tape or specially designed caps is a useful aid for the beginner as it cuts out the temptation of looking down at your hands. In this respect, a keyboard and monitor are better than a typewriter and paper. The screen is at eye level, and the temptation to glance downward is lessened. Errors can also be corrected immediately, ensuring greater accuracy, which is the next stage in mastering touch-typing.

Accuracy is a matter of practice and concentration. You must strike the key with a firm, quick tap, squarely in the middle of the key; hitting the keys with an equal degree of regularity, which builds up a rhythm that, in time, becomes natural. Eventually, using the wrong finger or striking the wrong key feels awkward, and this natural rhythm contributes greatly to building up speed.

Speed is measured by time tests and exercises, and is learned only after the first three stages we have discussed. Top touch-typists can reach speeds of over 100 words per minute (wpm). For the beginner, a reasonable speed to aim for is about 30 wpm.

Several different teaching manuals are available, and over the years a number of teaching methods have been devised. These still employ the principles we have outlined. Pitman, the international secretarial school, still produces a manual that was first published 35 years ago, as well as an updated version that teaches the keyboard through the typing of *words* from the very first exercise. Because small and commonly used words are used in these initial exercises, little effort is required to handle the spelling, which leaves the beginner free to concentrate on acquiring the necessary technique.

The Dico Typing Course claims to have taught

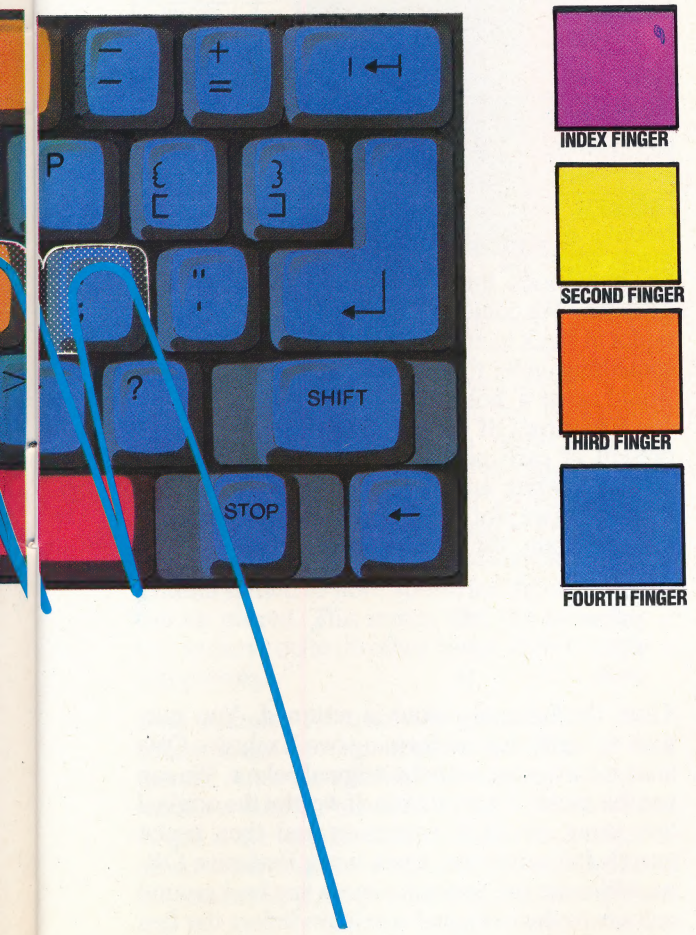
Home Base

This illustration shows the location of the home keys for the left and right hands and the allocation of keys for each finger. The index and fourth fingers are responsible for the largest range of keys because they have the greatest freedom of movement

KEVIN JONES

11-year-old children to touch-type in under 10 hours. This approach is based on the work of American psychologist BF Skinner. In this method, the beginner tears the page out of the manual and types out the answer beneath the exercise set. This follows the traditional method of teaching in which the student copies set exercises exactly. The exercises are accompanied by drawings of hands in the correct position on the keyboard.

One school to bring old-fashioned touch-typing methods directly in line with modern thinking is Sight and Sound, an organisation with 11 training centres in Britain and more abroad. Sight and Sound's teaching methods involve the use of flashing lights and recorded cassettes. This audio-visual teaching system simulates the responses of seeing, hearing and reacting simultaneously. On a large overhead board a light flashes a letter. The recorded tape synchronises with the board and the pre-recorded voice of the instructor calling 'Now!' will set the speed at which letters are to be typed. As you become more advanced, the speed at which the instructor will call for the letter to be typed will increase. Sight and Sound's technique has been described as 'brainwashing', and even the instructors will admit that they do not fully understand why this method works so effectively. However, the number of satisfied customers who learn to touch-type efficiently and effortlessly indicates that the system works, whatever the reason.



Software Roundup

Computer programs that teach touch-typing are either text- or games-based. Text-based packages rely heavily on exercises, repetition and drills. Most lessons in this sort of program are text-based, usually in the form of written exercises that the user must copy exactly. Games-based packages teach through the use of graphics, fast action and sound.

IAN McKINNELL

Type Invaders

Publisher: Carswell Computers
Price: Disk/£10.50, cassette/£6.95
Machine: BBC Model B

This is a games-based package for those with rudimentary typing skills. Its simple graphics display 'attacking' letters that must be destroyed by typing them correctly. Words and letters that have been missed attack again and can eventually blow up defence lines and occupy your land. There are 10 different levels of play, ranging from capital letters only to five-letter words incorporating upper and lower case, capitals and figures. There are also four separate speeds: easy, fast and rapid. Skilled typists will find even the 'rapid' option easy. However this is a fun, practical package for slow to average typists. Beginners should practice first with the Typeasy package from the same manufacturer

Sprintyper

Publisher: Micro Software International
Price Cassette/£14.95
Machine: Commodore VIc-20

This is a text-based package, which claims to promote speed and accuracy for both beginners and advanced typists. It has a library of 356,635 sentences to improve skills. To begin with, an easy sentence is show on the screen, to be typed as quickly as possible. A low tone signals a mistake, and will stop only when a correction has been made. Once the sentence has been copied correctly, the typing time, number of errors and a record time appears on the screen. Sprintyper is essentially a speed test, offering the beginner little in the way of constructive exercises

Typing Tutor II

Publisher: Microsoft
Price: Disk/£21.85
Machines: Apple IIe and Apple IIe+

To run this package you will need Applesoft in ROM, 48K of memory, a disk drive and DOS 3.3. This is a menu-driven text-based package, providing a combination of lessons, practice paragraphs and speed tests. Typing Tutor's most important feature is the 'Time Response Monitoring' system, which checks the typing 100 times per second, detecting even the slightest pause that occurs should the eyes move from the screen to the keyboard. Beginners start with a number of letters to practise. As they become familiar with these letters, and when the speed of typing is equivalent to 30 words per minute, those letters are transferred to a FAST column and new letters selected for practice. For experienced typists, the progress report on the practice paragraph details the number of errors made, the keys on which these errors were made, speed and accuracy. This is highly recommended for typists at all levels of skill. It is initially a little difficult to follow, and users are advised to study the documentation carefully beforehand

ADDING DAZZLE

At this stage in our programming project for the BBC Micro and the Electron, we have developed all the routines that form the basic skeleton of our Mines game. We can now concentrate on adding refinements to the program that will make the game visually appealing and exciting to play.

The first addition we shall make is a 'sniping' routine. This simulates a sniper firing across the minefield trying to hit either the mine detector or the assistant. The firing will be shown as a high resolution line crossing the screen from the left margin of the minefield to the right. To introduce a random element into the sniping, we shall select the co-ordinates of the starting and finishing points using the RND function. The values of xstart and xfinish are set in the initialise variables procedure. The difference between these two values is 1,024 graphics units. If the sniping line is to detect a hit on either the detector or the assistant it must draw a short segment of the line, then test the area ahead for the presence of logical colour 1 (using the POINT command) before drawing the next short segment. This sequence must be repeated until the other side of the screen is reached or a hit is made.

We must now decide on the step length we wish to use. If we choose a very short step length, then the time taken to draw the line will increase. If, however, we have too long a step length we may miss detecting the targets altogether. As each character cell is the equivalent of 64 graphics units across, a step length of half a character cell (i.e. 32 graphics units) would seem reasonable. Therefore, if we choose our step length in the x direction (dx) to be 32 units, we can draw the line in a total of 1024/32=32 steps. If we calculate the y co-ordinates of the start and finish points randomly, then the appropriate step length in the y direction (dy) can be calculated by dividing the difference between the two values by 32.

Our final problem is to find some way of erasing the line after it has been drawn. The solution lies in BBC BASIC's concept of logical colours and its ability to perform logical operations between them. In mode 5 there are four logical colours. Unless we modify them they are:

Logical Colour	0	1	2	3
Binary Equivalent	00	01	10	11
Normal Actual Colour	black	red	yellow	white

Using GCOL we can perform various logical operations between the colour we are plotting and the colour that is already there. The command has two parameters, the second of which indicates the logical colour to be plotted. The first number sets the method of plotting:

GCOL0	Plots the colour specified
GCOL1	Performs OR operation
GCOL2	Performs AND operation
GCOL3	Performs Exclusive OR operation
GCOL4	Performs NOT on the colour already there

This may sound complex, but a few examples should make the operation of the command clear. If white (logical colour 3) is present at the position that we wish to plot to and we want to plot red (logical colour 1) the various modes of operation of GCOL will produce the following results:

GCOL0,1	Will erase white and plot red	
GCOL1,1	ORs red and white to produce white	red 01 white OR 11 white 11
GCOL2,1	ANDs red and white to produce red	red 01 white AND 11 red 01
GCOL3,1	Exclusive ORs red and white to produce yellow	red 01 white EOR 11 yellow 10
GCOL4,1	NOTs white to produce black	white 11 black 00

So how does this help us with our erasing problem? We could plot the line in white and then replot in black to erase it. But if there were already something under the line, such as a mine, then this would cause a 'hole' to be left in it. However, we can Exclusive OR the red with the colour already present at each point the line crosses. When it crosses a white area, we shall get a yellow line segment. If we plot over the same area in Exclusive OR red again, the final result would be:

red	01
yellow	10
EOR	—
white	11

Thus, the original colour is returned. You may wish to verify that performing two Exclusive ORs always leaves you with the original colour. We can use this fact to erase our line. If we plot the original line using an EOR operation and then replot exactly the same line, again using Exclusive OR, we will erase the line and restore any background colours to their original condition before the first

plot. Here is the complete listing for the snipe procedure:

```
3110DEF PROCsnipe
3120ystart=RND(750)+220
3130yfinish=RND(750)+220
3140dx=32:dy=(yfinish-ystart)/32
3150GCOL 3,3
3160PROCline
3170IF POINT(x,y)=1 THEN PROCexplode(x,y) ELSE PROCline
3180ENDPROC
```

And this is the line procedure listing:

```
3450DEF PROCline
3460SOUND0,-8,4,5
3470x=xstart:y=ystart
3480MOVE x,y
3490REPEAT
3500DRAW x,y
3510x=x+dx:y=y+dy
3520UNTIL x>xfinish OR POINT(x,y)=1
3530ENDPROC
```

THREE ADDITIONAL FEATURES

As we saw in the last instalment, quite complicated sounds can be generated by the BBC Micro. For those of you with a musical bent, we shall now add a short tune to the program. To make things as simple as possible we shall only use one channel. The tune can be played by simply specifying the frequency and duration of each note in the tune.

```
4090DEF PROCmusic
4100REM ** 1ST BAR **
4110SOUND1,-8,213,5
4120SOUND1,-8,209,5
4130SOUND1,-8,213,5
4140SOUND1,-8,209,5
4150SOUND1,-8,213,5
4160SOUND1,-8,193,5
4170SOUND1,-8,205,5
4180SOUND1,-8,197,5
4190REM ** 2ND BAR **
4200SOUND1,-8,185,20
4210SOUND1,-8,165,5
4220SOUND1,-8,185,5
4230SOUND1,-8,193,20
4240REM ** 3RD BAR **
4250SOUND1,-8,165,5
4260SOUND1,-8,193,5
4270SOUND1,-8,197,20
4280ENDPROC
```

Title Page: We can use the ideas of Exclusive OR plotting and relative point plotting to produce an interesting title sequence. This procedure draws the word MINES using high resolution graphics. Every new line drawn in the word is plotted relative to the last, so we can position the entire word anywhere on the screen simply by specifying the start point. If we plot the word and then replot in Exclusive OR before moving up and repeating the action, we can make the word appear to float up the screen. GCOL0,129 sets the background colour to red. Performing a subsequent CLG colours the whole screen red. At the same time, we can also play the tune defined above by calling PROCmusic. The information held in PROCmusic is processed rather more quickly than it is played, so a buffer is used to store SOUND information until it can be played. This means that the processor is free to move on to do other things while the tune is still playing.

Skill Factors: To make the game a little more challenging, we can employ the idea of skill factors. After the title has been displayed we shall ask for a number between 0 and 9, which will be stored in the variable skill. This can then be used to increase the number of mines on the minefield and the rate of sniping across the area. The first of these can be done by making a small alteration to the setup procedure given previously (see page 405). Change lines 1930 and 1940 to:

```
1930factor=skill*3+30
1940PROClay_mines(factor)
```

In addition, when we relay the mines during the reset procedure, we must calculate the number of mines remaining by changing line 3950 to:

```
3950mines_left=factor-score/150
```

The full listing for the title page procedure is:

```
1300DEF PROCtitle_page
1310GCOL 0,129
1320CLG
1330GCOL 3,3
1340PROCmusic
1350Y=100:X=0
1360REPEAT
1370X=X+20:Y=Y+50
1380FOR I=1 TO 2
1390PROCmines
1400NEXT I
1410UNTIL Y>700
1420:
1430PROCmines
1440PRINTTAB(0,20)"Skill factor (0-9)?"
1450PROCmusic
1460REPEAT
1470skill=GET-48
1480UNTIL skill>-1 AND skill<10
1490ENDPROC
1500:
1510DEF PROCmines
1520PLOT4,X,Y
1530REM ** LETTER M **
1540PLOT1,0,200
1550PLOT1,80,-100
1560PLOT1,80,100
1570PLOT1,0,-200
1580REM ** LETTER I **
1590PLOT0,40,0
1600PLOT1,80,0
1610PLOT0,-40,0
1620PLOT1,0,200
1630PLOT0,-40,0
1640PLOT1,80,0
1650REM ** LETTER N **
1660PLOT0,40,-200
1670PLOT1,0,200
1680PLOT1,120,-200
1690PLOT1,0,200
1700REM ** LETTER E **
1710PLOT0,160,0
1720PLOT1,-120,0
1730PLOT1,0,-200
1740PLOT1,120,0
1750PLOT0,-40,100
1760PLOT1,-80,0
1770REM ** LETTER S **
1780PLOT0,280,60
1790PLOT1,0,40
1800PLOT1,-120,0
1810PLOT1,0,-100
1820PLOT1,120,0
1830PLOT1,0,-100
1840PLOT1,-120,0
1850PLOT1,0,40
1860ENDPROC
```

Up to this point we have been using a temporary calling program (given on page 394) to knit our procedures together, but now we have assembled all the procedures that are required for the main program loop of the game. Erase the temporary calling program (lines 10 to 70) and enter the following listing:

```
2020DEF PROCloop
2030REPEAT
2040PROCupdate_time
2050PROCtest_keyboard
2060rand=RND(50-skill)
2070IF rand=1 THEN PROCsnipe
2080 UNTIL TIME>12099 OR end_flag=1
2090ENDPROC
```

Our calling program can now be written. Enter these lines:

```
1060hi_score$="00000"
1110MODE5
1120REM ** TURN OFF CURSOR **
1130VDU23;8202;0;0;0;
1140PROCtitle_page
1150CLS
1160PROCsetup
1170:
1180PROCloop
```

In the next and final instalment of the course, we shall look at producing the end-of-game scenario and present a complete listing of our program.

DISASSEMBLER

A *disassembler* is a software program for converting machine code back into Assembly language. It will change a byte value into the three-letter mnemonic for the particular op-code that it represents (LDA, JMP, etc.), and, from the addressing mode specified by that particular byte, it will decide what operand is represented by the next one or two bytes, and print it in suitable form alongside the op-code.

Disassemblers are very useful when examining or modifying machine code written by other people. However, it is very important to realise that a disassembler cannot turn a piece of object code back into its original source code — i.e. with all the labels and symbols — because no record of these exists in the object code.

DMA

Direct memory access is a hardware technique that allows more than one device to share a common area of memory. Specifically, it allows a microprocessor to allocate an area of memory for this purpose so that another device can read the contents of that area without interrupting the operation of the micro. One application for this in microcomputing is graphics programming. If the video controller chip can read the contents of the screen RAM directly, instead of requiring each byte to be fed to it from the CPU, operation will be much more efficient.

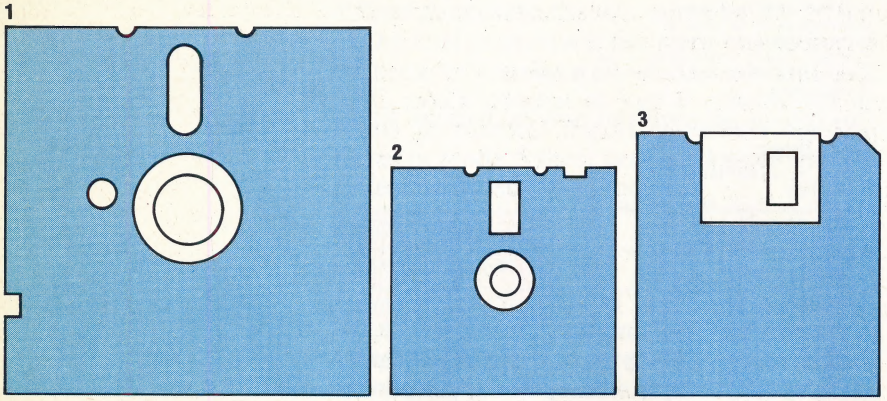
DMA works because the external device reads the memory in a different phase of the clock cycle from the CPU. The processor is thus completely 'unaware' that any other device is linked to the same area of RAM.

DOUBLE DENSITY

The capacity of a disk unit is determined by its recording density — that is to say, whether it records on one side of the disk or both. In the early days of microcomputing, there were two standard disk capacities: single density and *double density* — the latter featuring twice as many tracks on the same size disks. Double density disks required far more accurate control of the disk drive's read/write head, and initially were more expensive and less reliable.

However, disk technology has advanced a great deal in the last few years, and these original standards have less and less significance. A $5\frac{1}{4}$in disk can now store anything from 90 Kbytes to one Megabyte, and the new $3\frac{1}{2}$in drives, which can hold anything up to 700 Kbytes, are rapidly taking over the market.

Information Transfer
Double density refers to a method of transferring information onto the surface of disks, and is the same principle for all sizes of disk. We show double density disks in the following sizes: 1) $5\frac{1}{4}$in mini-floppy; 2) 3in microfloppy; 3) $3\frac{1}{2}$in microfloppy



DOUBLE PRECISION

Real numeric variables, the ones most commonly used in BASIC, generally store the equivalent of eight or nine decimal places ('equivalent' because the values are stored internally as binary, not decimal, numbers). These are called 'single precision variables'. *Double precision* variables store twice as many digits, and therefore are far more accurate.

Most programming languages for mainframe computers (particularly FORTRAN and ALGOL) give the programmer the option of using either single or double precision on each variable. Some BASICS now have this facility; using a symbol like # or ! after the variable name indicates double precision values, just as $ distinguishes string variables from numeric ones.

There are very few applications that require answers to be given to eight decimal places (the exceptions are fields like astronomy and codebreaking), so why the need for 16? The reason is that for every arithmetic function performed (addition, subtraction, etc.), there will be some loss of precision, because the least significant digit will be rounded up or down from the true result. In 'number crunching' applications, like engineering, statistics and weather forecasting, programmers must use considerable skill to prevent these errors from accumulating (thereby producing answers with little reliability). Double precision doesn't eliminate the problem, but it does help.

DOWNLOAD

Telesoftware is the name given to programs that can be transmitted from a central source to individual users. *Downloading* refers to the process of receiving the transmission and storing it in RAM or on disk. Originally, downloading was used to mean transferring a file from a central mainframe computer to a local intelligent terminal or minicomputer. Nowadays, you can download to a home computer over the telephone (the Prestel directory contains a large number of programs that can be purchased in this manner) or even over the airwaves — both television and radio networks have successfully transmitted programs on standard audio channels.

Downloading may radically alter the way that software is purchased in the future. Now that games programs rise and fall in popularity in weeks rather than months, holding large and expensive stocks is becoming a real problem for retailers. One idea under trial is the re-programmable cartridge, which the user can take back to the shop, where the program will be changed for a small fee. A special terminal downloads the program from a central source onto the cartridge, and then adds one to the 'popularity score' of that program.

LIZ DIXON

APPLE BITES BACK

Apple's Macintosh microcomputer is designed to make life easy for the novice computer user. Based on the technology developed for the more expensive Apple Lisa, the Macintosh, with its built-in disk drive, integral monitor, mouse and easy-to-use operating system, represents a major step forward in computer design.

The Macintosh is unlike any of the computers we have discussed so far in the course. In fact, it is unlike any other machine on the market. Although the Macintosh is primarily a business machine, Apple has chosen to create its own niche rather than follow the path taken by most other manufacturers who have adopted IBM standards in their machine designs. By taking this risky course, Apple has maintained its reputation as an innovator in an industry filled with 'lookalikes'.

The Macintosh comes in an unusual package. The sleek and slender system unit is small for a machine of its processing power. The display is a high resolution nine-inch screen, and the drive uses Sony $3\frac{1}{2}$in disks. There is a moulded handle on the cabinet casing, so the Macintosh can be classed as a truly portable machine. Together with the keyboard, mouse and an optional carrying case, the system weighs a total 11.6kg (25.6lb). The carrying case has compartments for all of the Macintosh's components, in the fashion of a picnic basket.

The Macintosh has a typewriter-style keyboard, which has an excellent 'feel' and is suitable for touch-typing. The keyboard has its own processor to handle special functions and international character sets. The other component of the 'Mac', as it is familiarly known, is the mouse. Named partly because of the 'tail' that connects it with the system unit, this hand-held device, the size of a cigarette packet, is moved around a flat level surface. A cursor makes corresponding movements on the screen, and can be used to select the activities the user wants the machine to perform. This is regarded as a much more 'user-friendly' approach to computer design than that employed by the majority of machines, which require a knowledge of specific operation commands. For example, if you wished to open a document file, you would manipulate the mouse so that the cursor fell on the small picture symbol (icon) representing a sheet of paper. A press of the button on the mouse would then open the screen for that activity. Having entered your file from the keyboard, the mouse would be used to return you to the main menu of icon commands, and the file could be saved to disk by placing the cursor over the symbol showing a disk.

IAN McKINNELL

Macintosh System
The Macintosh is designed to occupy as little space as possible on a desk-top. The extremely high resolution of the screen makes it possible to do graphics tricks usually seen only on machines costing 10 times the price

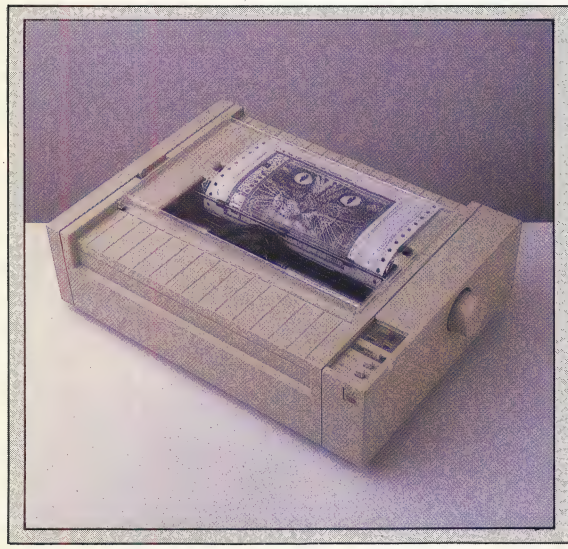
The Macintosh comes with 128 Kbytes of user memory, which can be increased to 512 Kbytes by replacing the existing RAM with 256 Kbyte chips. The Mac also has 64 Kbytes of ROM tightly packed with operating software, which handles virtually all of the system operations, as well as some special features. The Sony disk drive uses $3\frac{1}{2}$in disks, which store up to 400 Kbytes on one side and are more reliable than the $5\frac{1}{4}$in disks.

The Macintosh screen is 512 by 342 pixels, and is 'bit-mapped' so that each of its more than 175,000 points can be addressed individually. This

Signature Panel
Inside the Macintosh case, Apple has inscribed the signatures of the design team. Included are those of Steven Jobs and Steve Wozniak, who created Apple. Their genius made the Lisa and Macintosh computers possible

makes possible some truly stunning graphics applications. Besides being great fun, the graphics tricks of the Macintosh are very valuable to designers, architects, advisers, public relations people, photographers, and many others. As the Mac is designed to work specifically with Apple's high-speed ImageWriter printer, all of its impressive graphics will print out exactly as they appear on the screen.

Despite the high quality and reliability of the Macintosh hardware, it is the added strength of its software that makes the machine so exceptional. With the integration of hardware and software, and the extensive ROM-based operating commands, it is fairly easy for developers to transfer programs written for other computers to the Macintosh. The computer is so 'user-friendly' that it can literally be plugged in and put to work immediately without prior knowledge of computer operation.



Improving Image

The ImageWriter is a serial printer that generates text at a speed of up to 120 characters per second. Its speed is even more evident in graphics mode, because it generates graphics in bit-mapped fashion, following the format of the screen

Analogue Board
This board controls the video monitor and the power supply. There is no need for a fan on the Macintosh. Excess heat is channelled through metal plates to the vent slots in the cabinet

Built-in Speaker

Disk Drive Head

Screen Contrast Control

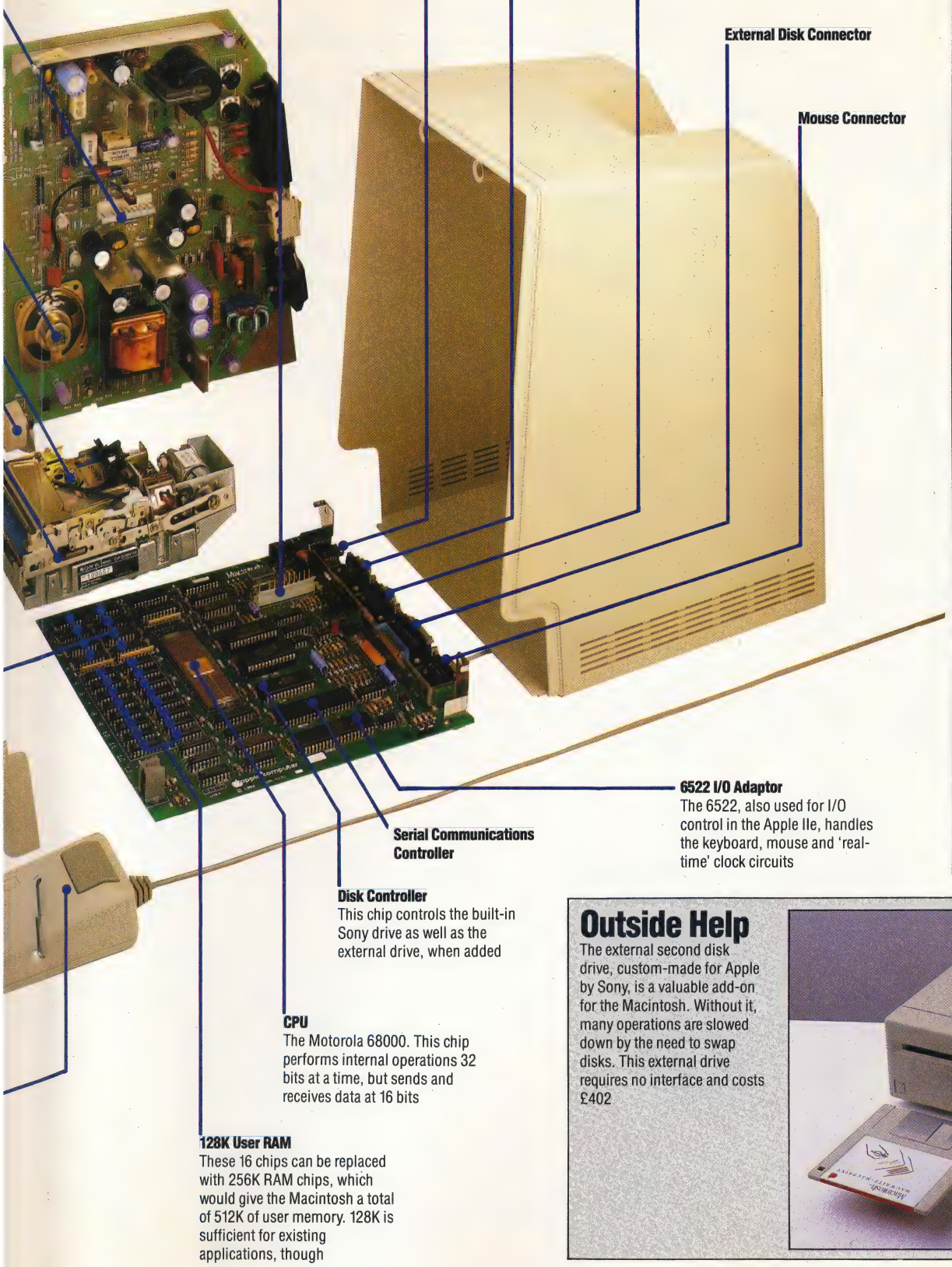
Sony 3½ in Disk Drive
Specially-built for Apple, this drive holds 400K on a side. Double-sided disks, when available, will hold 800K each

Video RAM
Some of the 22K required by the video display is drawn from these DMA (direct memory access) circuits

Keyboard
The Macintosh's detachable keyboard has its own processor to handle international character sets and special functions. No cursor keys are needed because of the mouse

Keyboard Connector

Mouse
The mouse controls movement of the cursor and is used to 'select' objects on the screen, then act on them according to instructions chosen from pull-down menus



Analogue To Digital Connection
A ribbon cable connects the analogue and digital boards

Audio Output

Printer Output

Serial Bus
Also referred to as 'virtual slots', the serial bus allows you to add a variety of peripheral devices

External Disk Connector

Mouse Connector

6522 I/O Adaptor
The 6522, also used for I/O control in the Apple IIe, handles the keyboard, mouse and 'real-time' clock circuits

Serial Communications Controller

Disk Controller
This chip controls the built-in Sony drive as well as the external drive, when added

CPU
The Motorola 68000. This chip performs internal operations 32 bits at a time, but sends and receives data at 16 bits

128K User RAM
These 16 chips can be replaced with 256K RAM chips, which would give the Macintosh a total of 512K of user memory. 128K is sufficient for existing applications, though

APPLE MACINTOSH

PRICE
£1,795 (excluding VAT)

DIMENSIONS
343×254×254mm (screen/disk drive cabinet)

CPU
Motorola 68000, 7.83 MHz

MEMORY
128K RAM, 64K ROM

SCREEN
Built-in monochrome monitor 512 × 342 pixels, windows, pull-down menus, 'icons'

INTERFACES
Mouse, printer, external disk drive, hi-fi amplifier, serial bus

LANGUAGES AVAILABLE
BASIC, COBOL, PASCAL

KEYBOARD
Typewriter-style, 59 keys, optional numeric pad

DOCUMENTATION
There is an operating manual with an audio cassette and 'guided tour' disk. Manuals are provided for MacPaint and MacWrite, which also include a combination disk and cassette guided tour.

STRENGTHS
The Mac has very powerful and easy-to-use software. The mouse makes operation very simple and straightforward. Streamlined component parts make servicing and updating very simple

WEAKNESSES
The Mac is a very expensive small computer and beyond the reach of many users. The single disk drive makes filing and disk operations slow and cumbersome. There is a lack of available software

Outside Help

The external second disk drive, custom-made for Apple by Sony, is a valuable add-on for the Macintosh. Without it, many operations are slowed down by the need to swap disks. This external drive requires no interface and costs £402

IAN McKINNELL

Apple invested heavily in the technology that created the Lisa and the Macintosh and is taking that technology to the public aggressively and proudly. In view of the quality of its design and construction, the Macintosh is clearly the harbinger of things to come.

Keyboard
The initial shipments of Macintosh machines for the UK market included the American keyboard. The Macintosh keyboard has a full international character set built in, so any version of the Macintosh can generate all of the special characters required for German, French, Spanish, Greek, and other languages

Desk-Top Window
The Macintosh screen follows the 'desk-top' analogy created for the Lisa computer, and uses 'windows' to show you portions of documents that are too large to fit on the screen. Windows can be layered one on top of the other, as shown here, so that using the Macintosh is very much like working on your desk-top

IAN McKINNELL

TROUBLED WATERS

Alligators, anacondas and mine-laying helicopters are just some of the hazards that must be negotiated as you take a trip up the river to rescue a group of stranded scientists in River Rescue. Originally developed for the Atari VCS video game console, River Rescue is now available on a range of popular home computers.

River Rescue is a pure shoot- em-up arcade game, with no pretensions towards being anything more. It is produced by Creative Sparks, Thorn EMI's software division, and the backing of such a large company is quite evident. It is available in versions for four home computers: the 48K Spectrum, Commodore 64, the Atari machines and the unexpanded Vic-20, and is supplied in a specially designed bubble pack instead of the usual music-cassette case.

Included with each version is a small instruction leaflet, which is extremely readable and helpful. This contains an invitation to join the Creative Sparks Software Club — membership is free, and benefits include introductory offers, news and competitions.

The game itself is basically very simple, but features enough action to satisfy any arcade addict. You control the river rescue power boat, and it is your job to rescue a group of scientists who are stuck in the upper reaches of the river. Why the scientists need rescuing in the first place is not explained, but the instructions tell you that you must take them to hospital, so presumably an accident of some type has occurred.

While attempting to pick up the injured scientists, you must steer your craft, which travels at considerable speed, around islands and logs, all the while blasting away at every alligator in sight. The Vic-20 version is a little different, with added hazards in the shape of anacondas and dug-out canoes. At intervals along the river bank you will see various jetties; it is from these that you must rescue the boffins. The successful transfer of a scientist to the other side of the river increases your score considerably, but you also score points by killing the alligators that infest the river.

Extra points are to be gained by transferring the scientists in groups, although your boat has a maximum capacity of nine scientists. This makes matters a bit more tricky, because all hands will be lost if your boat hits an obstacle. Therefore, you must choose whether to go for a high score and risk losing everything or play safe by transferring your passengers one at a time. To make matters worse, a helicopter — an aircraft in the Spectrum version — is likely to appear at any time and drop mines in the water, which must be blown up before you can proceed any further.

The Vic-20 version is conveniently supplied in cartridge format to avoid the tedium of cassette loading. Here, there is an option of either three or six stranded scientists and you have six lives per game. Furthermore, the points scored in each 'life' are carried over into subsequent incarnations, which makes things considerably easier. In this version, though, you have an extra three rivers to navigate.

River Rescue is an all-action, shoot-anything-that-moves type of game that has been carefully designed to make the play difficult enough to keep you occupied for some time, although it could be argued that it lacks the imagination necessary to make it really special.

River Rescue: For the 48K Spectrum, £6.95
For the Atari, £8.95
For the Commodore 64, £7.95
For the Vic-20 (unexpanded), £9.95
Publishers: Creative Sparks, 1st Floor, Thomson House, 296 Farnborough Road, Farnborough, Hants.
Author: Kevin Buckner
Joysticks: Kempston/Interface 1 (Spectrum)
Commodore-compatible joysticks (Vic-20 and Commodore 64))
Atari-compatible joysticks (Atari)
Format: Cassette;
Cartridge (Vic-20 only)

To The Rescue
River Rescue is seen here running on a Spectrum. The first photograph shows the title page, which gives a good indication of the graphics to come. The second photograph shows the game in progress. The boat is carrying one scientist to safety where he can join the others who have been rescued

LIZ HEANEY

CIRCULAR MEASURE

We look at 'recursion', a technique used in advanced programming such as artificial intelligence and the writing of compilers and assemblers. A functional knowledge of recursion can enhance a programmer's skills and add new dimensions to your BASIC programs. Our simple example, a Towers of Hanoi game, shows how easily the technique can be used.

The subject of this investigation is best summed up by a common joke definition:

Recursion: *see* Recursion

This circular definition demonstrates one essential feature of recursion — namely, something being defined in terms of itself. But it ignores another important feature: for recursion to be workable, there must be a way out of the circularity.

The puzzle we have used to illustrate recursion is The Towers of Hanoi. The puzzle begins with a pile of discs arranged in order of size, with the largest disc on the bottom of the pile and the smallest disc on top. To solve the puzzle, you must move all of the discs from the first pile to a second pile according to the following rules:

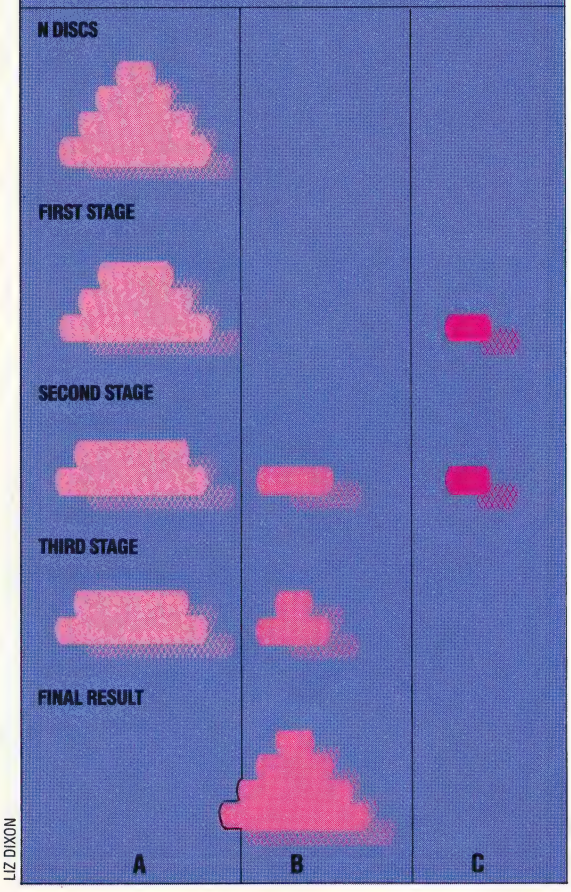
1) Only one disc may be moved at a time;
2) A disc may not be placed on a smaller disc;
3) There may never be more than three piles of discs.

The diagram illustrates how we utilise the concept of recursion to make the problem manageable. We begin with a pile of four discs. By assigning a variable N with the value of four, we indicate the total number of discs that must be moved. Since the rules do not allow the movement of more than one disc, we use a recursive formula to reduce the value of N by 1, then continue the calculation until N equals one. When N = 1, the program stops calculating and moves the appropriate disc.

If we are working with a version of BASIC that allows recursion, it is easy to write a program that follows the above process exactly. In the BBC BASIC program, all the work of calculating the moves is done in lines 1000 to 1050. The rest of the program is required to produce the moving pictorial display!

Tower Power

The program asks the player to input the number of discs to be moved by typing in the required number. The assigned number of discs are placed in pile A. The recursive algorithm then moves one disc at a time through piles A, B and C, in the first three stages shown below. Eventually, the original pile of N discs is reduced to zero and there are no more discs to move, as shown in the final result. When the movements have been completed, N discs will have been repositioned in sequential order in pile B



LIZ DIXON

THE SPECTRUM VERSION

To convert the Towers of Hanoi program to Spectrum BASIC we have to replace a recursive procedure with a recursive subroutine, which begins at line 1000 of our listing. Each time the subroutine has to make a recursive call to arrays M, A, B, or C, it increments the pointer variable J and puts the new variable values into M(J), A(J), B(J), and C(J). Subsequently, these new values can be used in the next call to the subroutine without disturbing the old values. At the end of the subroutine, the value of J is decremented, thus restoring the old values. This method can always be used for writing recursive subroutines in BASIC, no matter how complicated the recursion.

The display section of the program is straightforward, printing an object in a new position and erasing it by printing blank characters in the old position. The programs show the side view of a pile of discs. To make the piles look symmetrical, we have ended each odd-sized bar with graphics characters half made up of a space and half solid colour.

BBC Micro

```
 10 DIM M(10): DIM A(10): DIM B(10): DIM C(10)
 20 DIM D$(10,10): DIM H(3): DIM P(3,10)
 30 GO SUB 3000
 90 DIM M(100): DIM A(100): DIM B(100):
  DIM C(100)
100 INPUT "HOW MANY DISCS? ";N
110 IF N<1 OR N>10 THEN  GO TO 100
120 GO SUB 3100
130 LET J=1: LET M(J)=N: LET A(J)=1: LET B(J)
  =2: LET C(J)=3
140 GO SUB 1000
200 STOP
1000 IF M(J)=1 THEN  GO SUB 1500: RETURN
1010 LET J=J+1
1020 LET M(J)=M(J-1)-1
1030 LET A(J)=A(J-1)
1040 LET B(J)=C(J-1)
1050 LET C(J)=B(J-1)
1060 GO SUB 1000
1100 LET M(J)=1
1110 LET A(J)=A(J-1)
1120 LET B(J)=B(J-1)
1130 LET C(J)=C(J-1)
1140 GO SUB 1000
1200 LET M(J)=M(J-1)-1
1210  LET A(J)=C(J-1)
1220 LET B(J)=B(J-1)
1230 LET C(J)=A(J-1)
1240 GO SUB 1000
1300 LET J=J-1
1310 RETURN
1500 LET PA=A(J): LET PB=B(J)
1510 LET M$=D$(P(PA,N+1-H(PA)))
1520 FOR I=22-H(PA) TO 7 STEP -1
1530 PRINT AT I-1,10*(PA-1);M$;
1540 PRINT AT I,10*(PA-1);B$;
1550 NEXT I
1560 FOR I=10*(PA-1) TO 10*(PB-1) STEP SGN (PB-PA)
1570 PRINT AT 6,I;M$;
1575 PRINT AT 6,I;B$;
1580 NEXT I
1590 FOR I=6 TO 20-H(PB)
1600 PRINT AT I,10*(PB-1);B$;
1610 PRINT AT I+1,10*(PB-1);M$;
1620 NEXT I
1640 LET H(PB)=H(PB)+1: LET P(PB,N+1-H(PB))=P
  (PA,N+1-H(PA))
1650 LET P(PA,N+1-H(PA))=0: LET H(PA)=H(PA)-1
1660 RETURN
3000 LET B$="          ": LET C$=CHR$ 143+CHR$
  143+CHR$ 143+CHR$ 143
3010 LET C$="": FOR I=1 TO 10: LET C$=C$+CHR$
  143: NEXT I
3020 FOR I=1 TO 9 STEP 2
3030 LET D$(I)=B$( TO 4-INT (I/2))+CHR$ 133+C$
  ( TO 2*INT (I/2))+CHR$ 138+B$( TO 4-INT (I/2))
3040 LET D$(I+1)=B$( TO 4-INT (I/2))+C$( TO
  I+1)+B$( TO 4-INT (I/2))
3050 NEXT I
3060 RETURN
3100 INK 3: PAPER 6: BORDER 6: CLS
3120 FOR I=1 TO N
3130 PRINT AT 21-N+I,0;D$(I);
3135 LET P(1,I)=I: LET P(2,I)=0: LET P(3,I)=0
3140 NEXT I
3150 LET H(1)=N: LET H(2)=0: LET H(3)=0
3160 RETURN
```

Spectrum

```
 10DIM D$(12),H(3),P(3,12)
 20PROCINIT
100INPUT"HOW MANY DISCS (1-12)";N
110IF N<1 OR N>12 THEN 100
120PROCDISPLAY(N)
130PROCHANOI(N,1,2,3)
200END
1000DEFPROCHANOI(M,PA,PB,PC)
1010IF M=1 THEN PROCMOVE(PA,PB):
   ENDPROC
1020PROCHANOI(M-1,PA,PC,PB)
1030PROCHANOI(1,PA,PB,PC)
1040PROCHANOI(M-1,PC,PB,PA)
1050ENDPROC
1100DEFPROCMOVE(PA,PB)
1110D$=D$(P(PA,N+1-H(PA)))
1120FORI=24-H(PA) TO 10 STEP -1
1130PRINT TAB(13*(PA-1),I);B$;
1140PRINT TAB(13*(PA-1),I-1);D$;
1150NEXTI
1160FORI=13*(PA-1) TO 13*(PB-1)
          (PB-PA)
1170PRINT TAB(I,9);D$;
1180NEXT I
1190FORI=9 TO 22-H(PB)
1200PRINT TAB(13*(PB-1),I);B$
1210PRINT TAB(13*(PB-1),I+1);D$;
1220NEXT I
1240H(PB)=H(PB)+1:P(PB,N+1-H(PB))
   =P(PA,N+1-H(PA))
1250P(PA,N+1-H(PA))=0:H(PA)=H(PA)-1
1260ENDPROC
3000DEFPROCINIT
3020FOR I%=1 TO 11 STEP 2
3030D$(I%)=CHR$150+STRING$(5-
   I%DIV2," ")+CHR$234+STRING$(2*
   (I%DIV2),CHR$255)+CH R$53+STRING$
   (5-I%DIV2," ")
3040D$(I%+1)=CHR$150+STRING$(5-
   I%DIV2," ")+STRING$(I%+1,CHR$
   255)+STRING$(5-I%DIV2," ")
3050NEXT I%
3060B$=CHR$150+STRING$(12," ")
3070VDU 23,1,0;0;0;0;
3080ENDPROC
3100DEFPROCDISPLAY(N)
3110CLS
3120FORI=1 TO N
3130PRINT TAB(0,23-N+I);D$(I);
3135P(1,I)=I:P(2,I)=0:P(3,I)=0
3140NEXT I
3150H(1)=N:H(2)=0:H(3)=0
3160ENDPROC
```

Recurring Problems
This is a photograph of the Towers of Hanoi program running on a Spectrum. The colour of the blocks can be changed very easily. If you try to follow as the computer solves the problem, watch carefully - the action moves rather quickly!

IAN McKINNELL

TOP SECRET

Our course on program design has so far shown how programs may be constructed from small, largely independent units called modules. We have looked in detail at how such building blocks are designed, and here we show you how to use them in the development of a complete program.

When building a program, it is a good idea to develop an overall structure, consisting of a base level of general-purpose routines that are used by other routines of increasing specialisation on higher levels, all under the direction of a single control module at the top. This 'pyramid' structure will allow us to use a design method called 'program refinement' or 'top-down design'.

Top-down design, as its name suggests, entails designing the topmost control program first. We describe its functions in terms of calls to 'lower' level routines and, for the time being, we need not worry too much about how these lower-level modules will work. Once this is done, we move down a level and describe the workings of each routine called by the top-level module. Each routine is described in terms of the routines it must call, and this process is repeated level by level until we reach the lowest level. At that stage, the functions performed by the routine we are describing are so simple that they may be defined by using the programming language itself.

As an example, let us look at the design of a 'Hangman' game. Instead of the player trying to guess a word selected by the program, as is the case with most computer versions of the game, we want the program to guess a word that we have chosen. One way of achieving this, without giving the program a long list of English words, is to enter data on the likelihood of particular letter sequences occurring.

```
100 REM Initialise variables and arrays
                    .
                    .
                    .
500 REM *****Control Routine***********
510 REM
520 GOSUB 1000:REM Title & Help screens
530 GOSUB 2000:REM Set up Board
540 GOSUB 4000:REM Find word length
from player
550 GOSUB 8000:REM Select data set and
load it
560 GOSUB 3000:REM Guess a letter
570 GOSUB 4500:REM Check guess with
player
580 GOSUB 5000:REM Update the board
590 IF GAME_NOT_OVER THEN 560: REM
guess again until game is over
600 IF WIN THEN GOSUB 10000 ELSE
GOSUB 11000:REM Give appropriate
ending for win or lose
610 GOSUB 6000:REM ask the player for
another game
620 IF ANOTHER THEN 530:REM if
another then start again
630 GOSUB 7000:REM say goodbye and stop
640 END
```

We know before we start that certain things must be done: variables need to be initialised, arrays must be dimensioned, the 'board' display has to be set up and updated as necessary, and routines must be written that keep the score, that make guesses, and that end the game.

Our first attempt at designing the control routine has a simple REM statement to indicate that variables and arrays must be initialised – we can fill in all the necessary details at a later stage. The control routine itself is simply a pair of loops. The outer loop (line 620) tests to see whether the user is signalling the end of a session, while the inner loop (line 590) tests to see if the game has ended.

Should we need to test the control routine, we must set up dummy subroutines to match the GOSUBs. Each GOSUB in the control routine should have a REM statement to explain its function and should start at a convenient line number – preferably one that is a round figure, such as 1000 or 5000. It is a good idea to ensure that routines with similar functions are given standardised line numbers; this will make life easier when routines are moved from one program to another. For example, game instructions might be contained in a subroutine that begins at line 1000, while a GOSUB 7000 program line will always end a game by calling a standard routine.

Our initial control routine is kept short and simple. It will fit onto the screen and therefore is easier to understand and debug than a program that extends over several screens. The three variables, GAME NOT OVER, WIN and ANOTHER, are all flags that are set in the various subroutines called by the control routine and are used here to determine whether the control program works in the way we intend. It should be quite easy to spot any errors in logic in this simple control routine.

At this stage it is necessary to look at the program's structure with a critical eye – we need to ensure that the program behaves as it should in all circumstances. We can also start to make improvements in the program design; for example, we might like to make the instructions available at any stage of the game and it might also be a good idea to keep a record of how many games the computer or player has won and a list of words that beat the program. Any or all of these changes can be made at this stage.

The next step is to specify each of the

subroutines called by the control program. Our listings show how two of these routines might look. The first (beginning at line 4000) simply prompts the user for a number between 1 and 20 (the word length). It uses a general-purpose subroutine that is assumed to exist at line 51000, which will take a string specified in PROMPT$, print it and then accept a number input by the user. If this number is not an integer that falls between the limits set by MIN% and MAX%, an error message will be given and the user will be asked to input a new number. This subroutine may easily be used in other programs, and a library of such general-purpose modules may be built up for use in later projects.

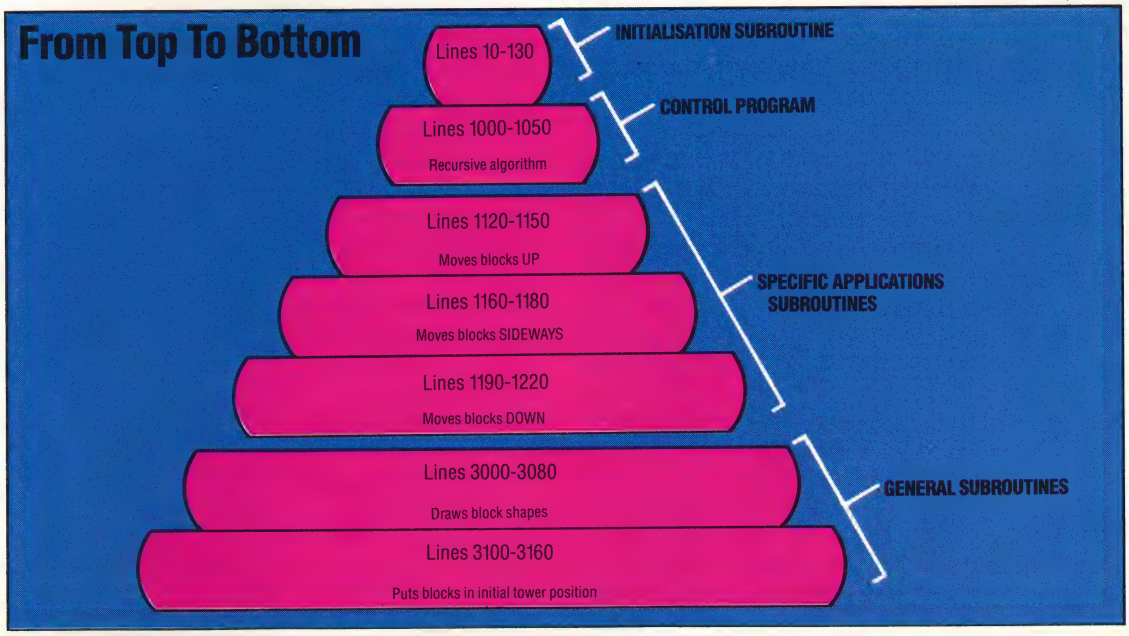
```
4000 REM Discover word length from
player
4010 REM
4020 PROMPT$="How many letters are
there in your word ?"
4030 MIN%=1
4040 MAX%=20
4050 GOSUB 51000:REM input an integer
between MIN% & MAX%
4060 WORDLEN%=RESP%:REM RESP% is used by
the subroutine at 51000 to pass back
the response
4070 RETURN

8000 REM select data set and load it
8010 REM
8020 IF WORDLEN%>7 THEN FILE_L%=8
                   ELSE FILE_L%=WORDLEN%
8030 FILENO_L$=STR$(FILE_L%)
8040 FILENAME$="TABLE"+FILENO_L$
8050 GOSUB 9000:REM OPEN, READ & CLOSE
the file with the likelihood data for
the appropriate word length.
8060 RETURN
```

The other routine (beginning at line 8000) uses local variables (FILE L% and FILENO L$). We have assumed that the data needed to guess a letter is in eight sets of tables that give the likelihood of finding any particular letter next to any other. As we want only one set of data in RAM at any time, we must build up a string in FILENAME$ to hold the name of the data file, and then call the subroutine at line 9000 to read the file.

In many cases, we will find that our program will move directly from one routine to another. However, we will usually want to create an extra routine that calls each of the other two in turn. This may seem like an unnecessary complication, but it allows us to keep a tight control over the program's 'flow' and it has the added bonus of keeping program modules separate so that they may be easily added to other programs.

This use of subroutines that are transportable from one program to another does involve extra work, and care must be taken when designing the routines so that they are suitable for use in a wide variety of circumstances. This may often be achieved simply by replacing constants with variables. It is important that all subroutines should be well documented. The documentation should specify the exact purposes of the routine, giving details of the variables used, the values expected as input and output, and any side-effects (moving the cursor position, changing the memory map, closing files, and so on).

A standard layout is also very helpful; you should make sure that all line numbers have a fixed interval, the titles and comments are restricted to a set number of lines at the beginning of the routine, and that RETURN is always on the last line. Be sure to note the first and last line number of each routine. When a library routine is required, make sure that the program has an appropriate gap in its line numbers and then MERGE the subroutine into the program. If your micro has no MERGE command, it may be possible to use a text editor to combine programs that have been SAVEd in ASCII format rather than the usual 'tokenised' form. If this is not possible, your library subroutines will need to be typed in each time they are used. However, the fact that they will not need to be redesigned should make the extra work worthwhile.



Top-down Programming
This diagram illustrates the principle of top-down programming. We have used the Towers of Hanoi program that appears on page 475. The line numbers in the diagram refer to the BBC listing.

The first layer of the structure represents the initialisation program, which must be completed before the rest of the program can be executed. The CONTROL PROGRAM in our diagram represents the recursive algorithm, which performs the calculations and calls the other subroutines as necessary. The SPECIFIC APPLICATIONS SUBROUTINES (lines 1120 to 1220), are used to move the block shapes from pile to pile in the display. The final two sections of the diagram, GENERAL SUBROUTINES, represent the last two sections of the program that are used to format the initial display and create the design for the blocks. Compare this structure with the listing, and you will see that the program is constructed in exactly this sequence

NOT SO FAST

We have often stated that the main advantage of machine code is the speed with which programs are executed. However, Assembly language programmers often find that their programs run too fast, and they need to insert time delays to slow them down. We look at the most popular methods for creating 6502 and Z80 software delays.

Delay loops can be implemented in 6502 Assembly language in several ways. The most obvious and simple method is to load one of the index registers with a value and decrement it within a loop until it reaches zero:

DELAY LOOP	TIME TAKEN FOR EACH OPERATION
LDY #$07	Two cycles
DEY	Two cycles
BNE LOOP	Two cycles (3 cycles if branch to same page; 4 cycles if branch to different page)

Each machine code instruction takes a particular number of clock cycles to execute. Information about these can usually be found with the descriptions of how the instructions operate. For example, the DEY instruction takes two cycles and LDY in immediate addressing mode also takes two cycles. As each cycle takes one microsecond (a millionth of a second), we can calculate the 'real time' taken to execute the delay loop. The total number of cycles can be calculated as follows:

1) The LDY #$07 instruction takes two cycles.
2) The program branches back seven times. Each time there is a branch back then the BNE operation takes three cycles: hence the DEY and BNE instructions take $(2+3)\times7 = 35$ cycles.
3) But the last BNE does not branch back and, therefore, takes only two cycles.

The total number of cycles is, therefore, $2+35-1 = 36$. The time taken to execute the delay is thus 36 microseconds.

There are several problems associated with using machine code delay loops to cause 'real time' delays (that is, delays that can be measured accurately in seconds or microseconds). The first, and most important, is that while a processor is executing a machine code program it regularly suspends this activity to service other parts of the system, such as scanning the keyboard, updating the internal clock, and so on. These breaks in program execution are known as 'interrupts', and two types of interrupt occur on the 6502 chip: NMI (non-maskable interrupt) and IRQ (interrupt request). The name given to the first type of interrupt implies that there is nothing that can be done to stop these interrupts occurring, but it is possible to stop IRQ interrupts that are not vital to the functioning of the processor.

IRQ interrupts can be masked by setting a particular bit in the processor status register to one. This is done by the instruction SEI. IRQ interrupts can be re-enabled by resetting the same bit using CLI. If we mask the IRQ interrupts before entering the delay loop, we can improve its accuracy. If a non-maskable interrupt occurs during execution then this will cause errors in the timing. Our original delay loop listing should be altered as follows to mask interrupts:

INSTRUCTION	FUNCTION	TIME TAKEN
SEI	Disable IRQ	Two cycles
LDY #$07		36 cycles
DEY		
BNE LOOP		
CLI	Re-enable IRQ	Two cycles

Masking the IRQs in this way adds another four cycles to the routine, which will now cause a total delay of 40 microseconds, assuming that no NMIs occur.

Another aspect of delay loops is that of 'resolution' — that is, how the time taken to execute a delay loop varies between one counter value and the next. In our example routine, we loaded the Y register with a value of seven, but if we had used a value of six instead, the delay time would have been 35 microseconds $(2 + 2 + (2+3)\times6 - 1 + 2)$. A value of five in the Y register would have taken 30 microseconds and so on to a minimum resolution of five microseconds.

We can 'fine-tune' our program (to give timings other than multiples of five) by placing NOP instructions outside the loop. An NOP instruction means the processor will perform 'No OPeration', and take two cycles to do it. If we wished to create a delay of 44 microseconds, for example, two NOP instructions could be added to our program before (or after) the loop:

SEI	Two cycles
LDY #$07	Two cycles
NOP	Two cycles
NOP	Two cycles
DEY BNE LOOP	34 cycles
CLI	Two cycles

This type of delay has an upper time limit determined by the maximum value of Y that can be used. As the Y index register is eight bits, this maximum value is 255. This gives an upper limit of 1,280 microseconds (2+2+(2+3)×255 -1+2), or approximately one thousandth of a second. This is a long time in microprocessor terms, but not in human terms. Occasionally, we will require longer time delays. Slight improvements in the time length can be made by adding NOP instructions *within* the loop. For example, adding one NOP instruction improves the maximum delay time to 1,790 microseconds (2 + 2 + (2+2+3)×255 —1 + 2).

For substantially longer delays we must devise another method. The two most common ways of producing long delays are to use a second loop nested around the first, or decrement a larger number, say a 16-bit word made up of two bytes from memory. For each of these methods you may wish to calculate the standard of resolution that can be obtained.

NESTED LOOP COUNTER

```
DELAY SEI
      LDX #$04  ;use X reg as outer loop counter
LOOP1 LDY #$FF  ;use Y reg as inner loop counter
LOOP2 DEY
      BNE LOOP2 ;end of inner loop
      DEX
      BNE LOOP1 ;end of outer loop
      CLI
```

The inner loop of the above program takes 1,276 microseconds (2 + (2+3)×255 —1) to execute. The outer loop controls the execution of the inner loop and performs a DEX and BNE four times. The total time for this delay can be calculated as:
2 + 2 + (1,276+2+3)×4 — 1 + 2 = 5,129 microseconds.

Z80 TIME DELAYS

Each Z80 machine code instruction takes a different amount of time to execute (measured in units called 'T states'), and the Z80 runs at different speeds on different machines. To calculate the real time taken by each instruction, the number of T states for the instruction is divided by the clock frequency of the micro. For example, an instruction that takes four T states to execute on a processor with a clock frequency of 2MHz is performed in two microseconds.

Rodnay Zak's *Programming the Z80* contains timings for all the Z80 instructions. These are the CPU clock speeds for the popular Z80-based machines: ZX81 (3.25MHz); the Spectrum (3.5MHz); Tandy TRS80/Video Genie (1.7MHz); and the Amstrad (4MHz).

To perform a very small time delay, the NOP instruction can be used. This instruction, on a 2MHz micro, will give a delay of two microseconds. A number of these can be used in succession, but longer delays can be achieved by calling dummy routines; for example, the following routine will give a delay of 27 T states:

```
CALL DELAY
RET
```

In this example, the CALL instruction takes 17 T states, and the RET instruction takes 10. Thus, with a processor running at 2MHz, the delay will be 13.5 microseconds. To extend this delay slightly, NOP instructions could be included at the beginning of the routine.

To achieve longer delays, a loop needs to be used. In the following example, a register is loaded with a value, which is then decremented within a loop. The routine gives a delay of 99 T states (or 49.5 microseconds at 2 MHz).

INSTRUCTION	TIME TAKEN (T STATES)
CALL DELAY	17
(DELAY LOOP)	
LD B,5	7
DEC B	4
JR NZ,LOOP	12 (or 7 if true)

The three instructions beginning with LD B,5 are the delay loop itself. As in a 6502 machine code routine, the total time length for this routine is varied according to the value loaded into the register. The total number of clock cycles it takes to perform this code can be expressed as:

$$C = 24 + (N \times 16) - 5$$

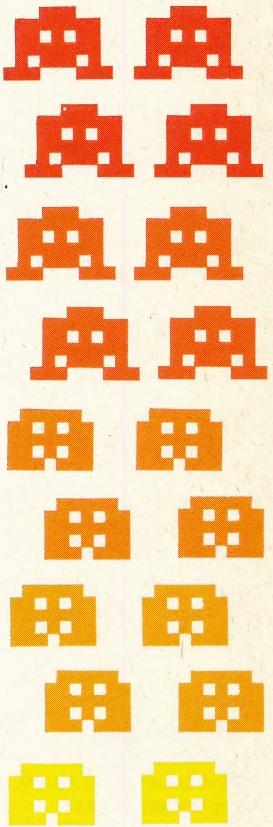
where N is the value loaded into the B register.

Nested loop counters can also be used. But here we must take other considerations into account. Firstly, any registers used during such a routine must first be 'pushed' to preserve their contents. Secondly, some machines have hardware interrupts that will upset the timing. The maskable interrupts are disabled and reinstated by the DI and EI instructions. The following routine makes use of nested loop counters:

INSTRUCTION	FUNCTION	TIME (T STATES)
DI	Disables interrupts	4
PUSH DE	Preserves register contents	11
LD D,nn	Value of inner counter	7
LD E,nn	Value of outer counter	7
CALL OLOOP	Jump to outer counter	17
POP DE	Restore contents	10
EI	Reinstate interrupts	4
:		
(OLOOP)		
DEC E	Decrement outer loop	4
RET Z	End if zero	11 or 5
(ILOOP)		
DEC D	Decrement inner loop	4
JP Z,OLOOP	Jump if D is zero	10
JP ILOOP	Else continue with inner loop	10

In this routine, the delay is increased if the value in the E register is increased. The routine will end when a decrement is made on the E register and the result is zero. Note that if the inner loop reaches zero, and the outer loop still has a value larger than one in it, the inner loop will be initialised to 255 and the inner loop will count down to zero before control is returned to the outer loop.

Timed Invasion
Machine code timing delays are necessary in games programs, particularly when there is a moving object on the screen that the player must interact with. A classic example of this is the Space Invaders game. Without timing delays, the movement of the invading aliens would be too fast. Through carefully-controlled timing delays, movement can be controlled as necessary to make the game play properly



COLOURFUL CONNECTIONS

Company Chairman
Richard Heath, the chairman who founded Prism as a subsidiary of ECC publications

Prism is a company that has grown from its simple beginnings as a distributor of Sinclair products to become a major marketing force. Instrumental in the development of Micronet, the company now markets Sinclair, Oric and Wren computers and is currently moving into the fast-developing home robotics field.

While most companies in the home computer industry are content to take a short-term view of the market by fulfilling immediate demand for hardware or software, Prism is looking to the future. A major hardware distributor, Prism played a leading role in the development of Micronet – the first large-scale database to be made available to home users – and is now involved in the distribution of low-cost robots.

The company was set up in 1982 by ECC Publications to develop Micronet, under the direction of Richard Heath and Bob Denton. Micronet uses Prestel, the largest public viewdata system in the country, to enable users of a wide range of home computers to download software, access information and exchange 'electronic mail' (see page 101). ECC Publications had already launched *Sinclair User* magazine, although at this time Sinclair products were available only by mail order or through the W H Smith retail chain. *Sinclair User* proved hugely successful, despite initial scepticism on the part of Terry Cartwright, now Prism's marketing director. 'I thought Sinclair was just a flash in the pan,' he admits, 'but we went to the first ZX Microfair with 8,000 subscription forms and there were queues of people right around the block and we handed out all the forms in just a few hours.'

Sinclair decided to move into the high street retail market, and Prism duly signed a contract to distribute the ZX81 and the newly launched Spectrum. In fact, the company name was deliberately chosen to foster an association with the Spectrum in the public mind – after all, if you direct a beam of light through a prism you'll end up with a spectrum of colours! Prism recently claimed to have sold over 500,000 Sinclair machines – an estimated 25 per cent of all UK home computer sales to date.

March 1983 saw the launch of Micronet by Prism, in partnership with British Telecom and Telemap. Prism took care of the hardware, distributing a range of modems (manufactured by O E Ltd and Thorn EMI) for the more popular machines. The most recent addition to this range was a modem for use with the Commodore 64.

Micronet now has around 10,000 subscribers, but Prism has recently sold its share of the network and is now concentrating on marketing and distributing computer hardware. In addition to the Spectrum, Prism now handles the Oric and Atmos machines as well as its 'own-brand' portable business machine, the Wren. After production delays of several months, the Wren, which is manufactured by Thorn EMI, is now appearing in the shops at a price of around £1,000.

Prism is also moving into a new area – the distribution of home robots. Interest in this field is growing fast, and Prism now markets 'Topo', a £1,500 robot imported from the USA, as well as a number of cheap robot kits selling under the name 'Movits' at prices between £10 and £35.

Terry Cartwright sees robots as an area of great expansion. 'There is a tremendous interest in robots,' he says. 'I don't know what people will do with them, but in 1976 nobody knew what would happen with Apple computers, either.' The company also hopes to begin distribution in September this year of the Sinclair QL. Cartwright expects Prism's diversification to continue in the future. 'Overseas expansion is very much a priority in the next 12 months,' he says.

Subscription Bonus
The company plans to rent these modems to customers, together with a year's subscription to Prestel and Micronet

Early Bird
The Wren, Prism's portable Z80-based business machine, comes fitted with two disk drives and a built-in modem to connect it to Prestel

THE HOME COMPUTER ADVANCED COURSE

INDEX TO ISSUES 13 TO 24

THE HOME COMPUTER ADVANCED COURSE

INDEX TO ISSUES 13 TO 24